Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

Отчёт по лабораторной работе

**Разбор и вычисление арифметических выражений**

Выполнил:

студент ИТММ гр. 381908-3

Минеев Даниил Сергеевич

Проверил:

Усова М.А.

Нижний Новгород

2019 г.

**Содержание**

[Введение 3](#_30j0zll)

[Постановка задачи 4](#_1fob9te)

[Руководство пользователя 5](#_3znysh7)

[Руководство программиста 6](#_2et92p0)

[Описание структуры программы 6](#_tyjcwt)

[Описание методов 6](#_3dy6vkm)

[Описание структур данных 8](#_1t3h5sf)

[Описание алгоритмов 8](#_4d34og8)

[Заключение 11](#_17dp8vu)

[Литература 12](#_3rdcrjn)

[Приложение 13](#_26in1rg)

[Приложение 1. Текст программы алгоритмов 13](#_lnxbz9)

# Введение

Как будущему программисту мне будет очень полезно понять структуру алгоритмов вычисления арифметических выражений и попрактиковаться в их написании на практике. Освоив эти навыки, я смогу решать более сложные задачи на основе этих стандартных навыков.

Несомненно, арифметические выражения действительно встречаются повсеместно. Алгоритмы передачи просты, элегантны и по-своему универсальны. Их знание может быть хорошей услугой для любого программиста.

# Постановка задачи

Цель: реализовать класс для разбора и вычисления арифметических выражений.

Требования к арифметическому выражению:

* может содержать скобки (), по желанию добавить поддержку разного вида скобок [], {}, скобки не поддерживаемые программой должны вызывать исключения;
* может содержать константы и символьные переменные (строчные буквы латинского алфавита);
* поддерживаемые операции: сложение ( + ), вычитание ( ̶̶ ), умножение ( \* ), деление ( / ), возведение в степень ( ^ ),
* операция взятия модуля ( | | );

Требования к программе:

Необходимо реализовать класс, который:

1. Принимает на вход строку, содержащую арифметическое выражение;
2. Выполняет её разбор, выводит сообщение об ошибке в случае некорректного задания выражения;
3. Выполняет вычисление значения выражения при заданных значениях переменных и выводит результат.

# Руководство пользователя

Данная программа написана с помощью программы MicrosoftVisualStudio 2017 на языке C++.

Класс предоставляет пользователю 3 метода:

1. char\* ConvertToPolish(char\* expression) - который принимает на вход строку выражения, переводит выражение в обратную польскую запись и возвращает строку выражения в этой записи.
2. void AddVar(char variable, double value) -с помощью этого метода можно

установить значения в переменные выражения.

1. double Calculate(char\* polishrecord) - метод считает значение выражения в обратной польской записи и возвращает

# Руководство программиста

Данная программа реализует следующий набор алгоритмов:

* Перевод в обратную польскую запись;
* Вычисление выражения, записанного в обратной польской записи;

И следующий набор опций:

* Проверка вводимых данных;
* Проверка последовательности скобок;
* Проверка последовательности операндов и операций;

## Описание структуры программы

В ходе работы программы на вход ей приходит строка с арифметическим выражением, которое она проверяет на корректность, затем вычисляет.

## Описание методов

char\* ConvertToPolish(char\* expression); - перевод арифметического выражения, из обычной записи, в обратную польскую.

double Calculate(char\* polishrecord); - выполняет арифметические действия с выражением.

int priority(char operation); - расчет приоритета, той или иной операции.

## Описание алгоритмов[[1]](#footnote-0)

#### Перевод в обратную польскую запись

Рассматриваем поочередно каждый символ:

* Если этот символ - число (или переменная), то просто помещаем его в выходную строку.
* Если символ - знак операции (+, -, \*, / ), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет (допустим он равен 3). Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.

Получив один из этих символов, мы должны проверить стек:

1. Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.
2. Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

Если текущий символ - открывающая скобка, то помещаем ее в стек.  
Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Рассмотрим алгоритм на примере простейшего выражения:

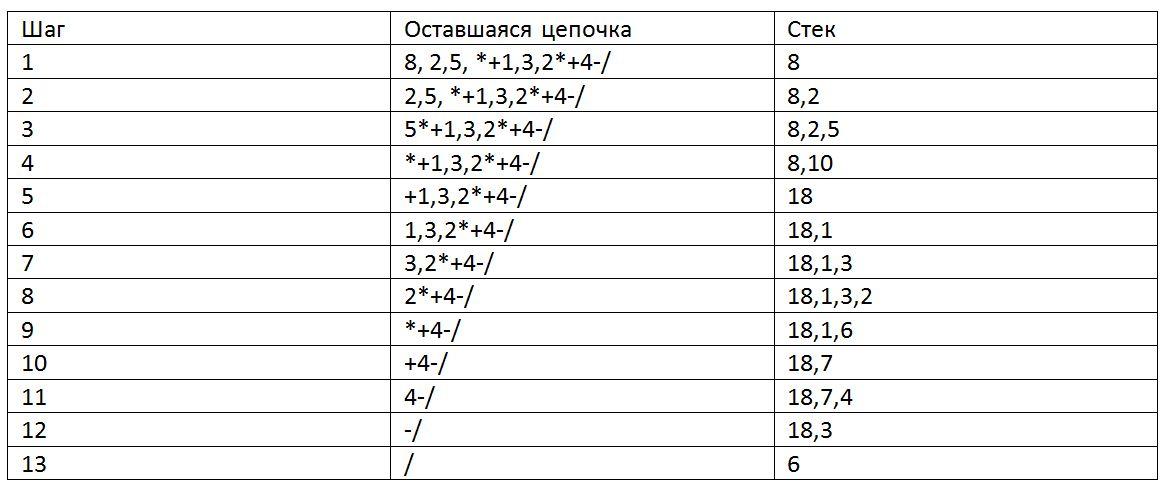
Дано выражение: a + ( b - c ) \* d

Рассмотрим поочередно все символы:

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Действие | Состояние выходной строки после совершенного действия | Состояние стека после совершенного действия |
| a | 'a' - переменная. Помещаем ее в выходную строку | a | пуст |
| + | '+' - знак операции. Помещаем его в стек (поскольку стек пуст, приоритеты можно не проверять) | a | + |
| ( | '(' - открывающая скобка. Помещаем в стек. | a | + ( |
| b | 'b' - переменная. Помещаем ее в выходную строку | a b | + ( |
| - | '-' - знак операции, который имеет приоритет 2. Проверяем стек: на вершине находится символ '(', приоритет которого равен 1. Следовательно мы должны просто поместить текущий символ '-' в стек. | a b | + ( - |
| c | 'c' - переменная. Помещаем ее в выходную строку | a b c | + ( - |
| ) | ')' - закрывающая скобка. Извлекаем из стека в выходную строку все символы, пока не встретим открывающую скобку. Затем уничтожаем обе скобки. | a b c - | + |
| \* | '\*' - знак операции, который имеет приоритет 3. Проверяем стек: на вершине находится символ '+', приоритет которого равен 2, т.е. меньший, чем приоритет текущего символа '\*'. Следовательно мы должны просто поместить текущий символ '\*' в стек. | a b c - | + \* |
| d | 'd' - переменная. Помещаем ее в выходную строку | a b c - d | + \* |

#### Алгоритм подсчета выражения в обратной польской записи

Обратная польская запись идеально подходит для вычисления формул на компьютере со стеком. Формула состоит из n символов, каждый из которых является либо операндом, либо оператором. Алгоритм для вычисления формулы в обратной польской записи с использованием стека прост. Нужно просто прочитать обратную польскую запись слева направо. Если встречается операнд, его нужно пометить в стек. Если встречается оператор, нужно выполнить заданную им операцию.  
В качестве примера рассмотрим вычисление следующего выражения: (8+2\*5)/(1+3\*2-4). Соответствующая формула в обратной польской записи выглядит так: 825\*+132\*+4-/  
Число на вершине стека – это правый операнд (а не левый). Это очень важно дл операций деления, вычитания и возведения в степень, поскольку порядок следования операндов в данном случае имеет значение (в отличие от операций сложения и умножения). Другими словами, операция деления действует следующим образом: сначала в стек помещается числитель, потом знаменатель, и тогда операция даёт правильный результат. Отметим, что преобразовать обратную польскую запись в машинный код очень легко: нужно просто двигаться по формуле в обратной польской записи, записывая по одной команде для каждого символа. Если символ является константой или переменной, нужно вписывать команду помещения этой константы или переменной в стек, если символ является оператором, нужно вписывать команду выполнения это операции.



**Рисунок 1 Пример работы алгоритма**

# Заключение

Во время лабораторной работы я изучал алгоритмы анализа и подсчета арифметических выражений. Я разработал и протестировал класс для проверки и оценки арифметических выражений строк.

В результате я получил ценные знания об алгоритмах перевода символьных строк в обратную польскую нотацию, подсчета выражений в обратной польской записи, а также умение применять их на практике, что, несомненно, поможет мне решать более сложные задачи.

# Литература

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн К.. Алгоритмы: построение и анализ. — 2-е изд. — М.: Вильямс, 2005. — С.1296.
2. Ахо А. В., Хопкрофт Д. Э., Ульман Д. Д.. Структуры данных и алгоритмы. — М.: Вильямс, 2000. — С. 231.
3. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск. — 2-е изд. — М.: Вильямс, 2001. — С.800.

# Приложение

### Приложение 1. Текст программы алгоритмов

PolishConvertor::~PolishConvertor() {

}

PolishConvertor::PolishConvertor() {

}

char\* PolishConvertor::ConvertToPolish(char\* expression) {

TStack operation\_stack;

char\* result = new char[MAX\_EXPRESSION];

int result\_size = 0;

result[0] = '\0';

char curr\_symbol;

bool prev\_sybol\_is\_blank = false;

for (int i=0;i<MAX\_EXPRESSION-1; i++) {

curr\_symbol = expression[i];

if (curr\_symbol == '\0')

{

while (operation\_stack.IsEmpty()!=1)

{

char operation = operation\_stack.Get();

if (operation != '(' && operation != '[') {

result\_size++;

result[result\_size - 1] = ' ';

result\_size++;

result[result\_size - 1] = operation;

}

}

result[result\_size] = '\0';

return result;

}

if (operation\_stack.IsEmpty()==1) {

if (priority(curr\_symbol)<DIGIT && curr\_symbol!=')' && curr\_symbol != ']') {

operation\_stack.Put(curr\_symbol);

}

else {

if (curr\_symbol != ' ')

{

result\_size++;

result[result\_size - 1] = curr\_symbol;

continue;

}

}

}

else {

if (priority(curr\_symbol) < DIGIT && operation\_stack.IsEmpty()!=1) {

char last\_operation = operation\_stack.Get();

int priority\_curr\_operation = priority(curr\_symbol);

int priority\_last\_operation = priority(last\_operation);

if (priority\_curr\_operation <= priority\_last\_operation) {

if (prev\_sybol\_is\_blank != true)

{

result\_size++;

result[result\_size - 1] = ' ';

}

bool direction\_to\_output = false;

if (last\_operation != '(' && last\_operation != ')' && last\_operation != '[')

{

result\_size++;

result[result\_size - 1] = last\_operation;

direction\_to\_output = true;

}

if (last\_operation != ']' && direction\_to\_output != true)

{

operation\_stack.Put(last\_operation);

}

if (curr\_symbol != ')')

{

operation\_stack.Put(curr\_symbol);

}

continue;

}

else

{

if (prev\_sybol\_is\_blank != true)

{

result\_size++;

result[result\_size - 1] = ' ';

}

result\_size++;

result[result\_size - 1] = last\_operation;

while (operation\_stack.IsEmpty()!=1)

{

char operation = operation\_stack.Get();

if (operation != '(')

{

result\_size++;

result[result\_size - 1] = ' ';

result\_size++;

result[result\_size - 1] = operation;

}

}

if (curr\_symbol != ')')operation\_stack.Put(curr\_symbol);

continue;

}

}

else {

if (curr\_symbol == ' ' && prev\_sybol\_is\_blank == true) continue;

result\_size++;

result[result\_size - 1]=curr\_symbol;

if (curr\_symbol == ' ') prev\_sybol\_is\_blank = true;

else prev\_sybol\_is\_blank = false;

continue;

}

}

}

return result;

}

void PolishConvertor::AddVar(char variable, double value) {

}

double PolishConvertor::Calculate(char\* polishrecord) {

TStack master\_stack;

int polishrecord\_size = strlen(polishrecord);

char curr\_symbol = polishrecord[0];

for (int i = polishrecord\_size-1; i >= 0; i--)

{

curr\_symbol = polishrecord[i];

master\_stack.Put(curr\_symbol);

}

double result = 0;

char operands[MAX\_EXPRESSION];

operands[0] = '\0';

int operands\_size = 0;

while (master\_stack.IsEmpty() != 1)

{

curr\_symbol = master\_stack.Get();

while (priority(curr\_symbol) == DIGIT)

{

operands\_size++;

operands[operands\_size - 1] = curr\_symbol;

if(master\_stack.IsEmpty() != 1)curr\_symbol = master\_stack.Get();

else

{

return result;

}

}

if(operands[operands\_size-1]==' ')operands[operands\_size-1] = '\0';

else operands[operands\_size] = '\0';

float operand1 = 0;

float operand2 = 0;

char blank[MAX\_EXPRESSION];

char result\_string[MAX\_EXPRESSION];

if (curr\_symbol == '+')

{

int err=sscanf(operands, "%f %f", &operand1, &operand2);

if (err != 2)throw std::logic\_error("Error expression format!");

result = operand1 + operand2;

}

else if(curr\_symbol=='-')

{

int err = sscanf(operands, "%f %f", &operand1, &operand2);

if (err != 2)throw std::logic\_error("Error expression format!");

result = operand1 - operand2;

}

else if (curr\_symbol == '/')

{

int err = sscanf(operands, "%f %f", &operand1, &operand2);

if (err != 2)throw std::logic\_error("Error expression format!");

result = operand1 / operand2;

}

else if (curr\_symbol == '\*')

{

int err = sscanf(operands, "%f %f", &operand1, &operand2);

if (err != 2)throw std::logic\_error("Error expression format!");

result = operand1 \* operand2;

}

else if (curr\_symbol == '^')

{

int err = sscanf(operands, "%f %f", &operand1, &operand2);

if (err != 2)throw std::logic\_error("Error expression format!");

result = pow(operand1, operand2);

}

else if (curr\_symbol == ']')

{

int err = sscanf(operands, "%f", &operand1);

if (err != 1)throw std::logic\_error("Error expression format!");

result = abs(operand1);

}

else if (curr\_symbol == '\_')

{

int err = sscanf(operands, "%f", &operand1);

if (err != 1)throw std::logic\_error("Error expression format!");

result = operand1\*(-1);

}

sprintf(result\_string, "%f", result);

int res\_sym\_counter = strlen(result\_string);

for (int j = res\_sym\_counter-1; j >= 0; j--)

{

master\_stack.Put(result\_string[j]);

}

operands[0] = '\0';

operands\_size = 0;

}

return result;

}

int PolishConvertor::priority(char operation) {

char priority\_3[] = { '(',')','[', ']', '\_' };

char priority\_2[] = { '^'};

char priority\_1[] = {'\*','/'};

char priority\_0[] = { '+','-' };

int priority3\_size = 5;

int priority2\_size = 1;

int priority1\_size = 2;

int priority0\_size = 2;

for (int i = 0; i < priority3\_size; i++)

{

if (operation == priority\_3[i]){

return 3;

}

}

for (int i = 0; i < priority2\_size; i++)

{

if (operation == priority\_2[i]) {

return 2;

}

}

for (int i = 0; i < priority1\_size; i++)

{

if (operation == priority\_1[i]) {

return 1;

}

}

for (int i = 0; i < priority0\_size; i++)

{

if (operation == priority\_0[i]) {

return 0;

}

}

return DIGIT;

}

1. Программную реализацию см. в Приложении [↑](#footnote-ref-0)